Worksheet 3 Bubble sort and insertion sort
Answers

Task 1

1. Use the girls' name cards for Exercise 1.
   Place the cards in ascending order of popularity, i.e.
   Sophie, Lily, Jessica, Isabella, Ava, Poppy, Emily, Isla, Olivia, Amelia
   Perform a manual bubble sort to get the cards in alphabetical order.

   What are the last 2 names after the first pass? Amelia, Sophie

   What are the first two names after the second pass? Jessica, Isabella

   What are the third and fourth names after the third pass? Jessica, Emily

   What are the seventh and eighth names after the fourth pass? Lily, Olivia

   What are the fourth and fifth names after the fifth pass? Isla, Amelia

   What are the first two names after the sixth pass? Ava, Emily

   Which names are out of sequence after the seventh pass? Ava, Amelia and Emily

   Are any names out of sequence after the eighth pass? Yes – Ava and Amelia

   How many passes were needed to sort the cards? 9

2. Complete the bubble sort algorithm given below.

```
for i = 0 to n - 2
  for j = 0 to (n – i - 2)
      if names [j] > names[j + 1]
          temp = names[j]
          names[j] = names[j + 1]
          names[j+1] = temp
      endif
    next j
next i
```

See Python program bubble sort Girls Names.py

Task 2

3. Place the name cards in ascending order of popularity:

   Sophie, Lily, Jessica, Isabella, Ava, Poppy, Emily, Isla, Olivia, Amelia

   Perform a manual insertion sort on the cards to put them into alphabetical order.

   What sequence are the first four cards in after 4 moves have been made?

   Ava, Isabella, Jessica, Lily

4. The following numbers are to be sorted into ascending order using an insertion sort:

   15, 73, 29, 66, 35, 11, 43, 21

   (a) Show the sequence of the numbers after each pass through the insertion sort algorithm

```
aList = [15, 73, 29, 66, 35, 11, 43, 21]
#assume first element of array is aList[0]
for j = 1 to len(aList) - 1
   nextNum = aList[j]
   i = j – 1
   while i >= 0 and aList[i] > nextNum
      aList[i + 1] = aList[i]
         i = i - 1
   endwhile
   aList[i + 1] = nextNum
next j
```

15	**73**	**29**	**66**	**35**	**11**	**43**	**21**
15	**73**	**29**	**66**	**35**	**11**	**43**	**21**
15	**29**	**73**	**66**	**35**	**11**	**43**	**21**
15	**29**	**66**	**73**	**35**	**11**	**43**	**21**
15	**29**	**35**	**66**	**73**	**11**	**43**	**21**
11	**15**	**29**	**35**	**66**	**73**	**43**	**21**
11	**15**	**29**	**35**	**43**	**66**	**73**	**21**

11	**15**	**21**	**29**	**35**	**43**	**66**	**73**

(b) How many passes are made through the data for a dataset of n numbers? n - 1

See Python program insertion sort.py

Extension task

5. Write an algorithm which will compare the length of time to sort a list of n random numbers using a bubble sort and an insertion sort.

   If you have time, write the program in a language of your choice and experiment with different values of n. Be prepared to wait a long time if you choose a number greater than 10,000!

   Which is fastest for 10 numbers?
   Which is fastest for 100 numbers? 1000 numbers?

   See Python program timing sorts.py

   The algorithm has a main program which asks the user how many numbers (n) they want to sort, generates n random numbers and puts them in 3 identical arrays numbers1, numbers2 and numbers3.
   The main program then gets the clock time, calls the first sort, gets the clock time again and prints the elapsed time. It does this again for the other two sorts.

   For a very small array of 10 numbers, the bubble sort and insertion sort are faster than the merge sort. For 10,000 numbers, however, the merge sort takes only a small fraction of the time taken by the other sorts.

   If you run the program with the same number of values each time, the timing will be different because the computer is multiprocessing and it depends what other tasks it is carrying out. A smaller effect is how well sorted the initial list of numbers is, but as these are random numbers, this will be negligible.